CHAPTER

11

SYSTEM LOGGING

After reading this chapter and completing the exercises, you will be able to:

- Explain the purpose of system log and other log files
- View the boot messages on your Linux system
- Configure the system logging daemons
- Maintain log files

In the previous chapter you learned about managing Linux resources (such as memory and CPU time) to keep processes running efficiently for all users. You learned about some command-line and graphical utilities that help you monitor the status of memory and all of the processes running on Linux. You also learned to control how individual processes use CPU time in Linux.

In this chapter you will learn how programs write messages to special Linux files, known as system log files, to help you track the activities of those programs. You will learn why log files are important, how to configure them to meet your needs, and how to maintain the log files to help keep your Linux system secure and running smoothly.

INTRODUCING SYSTEM LOGS

On any ship, the captain keeps a log of information about each day, including where the ship has traveled, its cargo, and any noteworthy events. The log serves as a record not only for the captain and crew, but also for others who may need detailed information about the ship.

In much the same way, Linux keeps detailed records of events within the system. These records, known as **log files**, are created by many programs. As the system administrator, you can refer to the log files to determine the status of your system, watch for intruders, or look

for data about a particular program or event. Table 11-1 lists some commonly logged events and the location of the corresponding log files. This chapter describes the most important log files for general system administration.

Table 11-1 Commonly Logged Events and Their Log Files

Event	Path and filename of the log
Main system messages	`/var/log/messages`
Web server transfers	`/var/log/httpd/access_log`
FTP server transfers	`/var/log/xferlog`
E-mail server information	`/var/log/maillog`
Automatic script executions	`/var/log/cron`

The first log file in Table 11-1, **`messages`**, contains messages produced by the Linux kernel and most of the key programs running on the Linux system. It is stored in the same location on all Linux systems—in the `/var/log` directory. On your system, the other log files may have different default locations than those listed in Table 11-1. For example, Red Hat Linux stores the Web server transfer log (`access_log`) in the directory `/var/log/httpd`. Caldera OpenLinux stores the same file in the directory `/var/log/httpd/apache`. Later in this chapter you will learn how to configure log files to suit your own needs and preferences.

The Purpose of Linux Log Files

On any Linux system, many events go on in the background as users log in and do their work. **Daemons** are special-purpose background processes designed to watch for network activity, run other programs, and monitor user actions. The status information collected by daemons is not displayed on the screen. Instead, it is written to log files, which you can then review whenever you choose. Among other things, log files allow a system administrator to:

- Check for potential security problems, such as repeated login failures or a program that is stopped and restarted without the knowledge of the system administrator
- Review what was happening on the system in the moments before a major problem occurred
- Manage the system load by computing statistics based on the log file information

In the next section you will learn about the `messages` file. This is the main log file, where log messages from the Linux kernel and many important Linux programs are stored. For example, the `messages` file contains a record of login attempts, FTP server connections, and each occurrence of a daemon being stopped or started.

The `messages` File

The main system log for Linux is stored in the file `/var/log/messages`. Many different programs write messages to this file. A **message** is a description of what is happening within a program. The message may report information (someone has logged in), a warning (someone tried to log in unsuccessfully), or a serious error indicating that a program is about to crash.

Several sample messages are shown later in this section. A number of daemons, like the Web server, e-mail server, and login security programs, write to the file, as does the Linux kernel itself. The messages from the kernel tell you about low-level system activities such as when devices are first initialized and when daemons are started by the kernel.

The `messages` file uses a standard format. Each line of the file makes up an individual log message. Each message, in turn, contains the following information:

- The date and time when the event being logged occurred (often called the **timestamp**)
- The hostname (or computer name) of the system on which the event occurred
- The name of the program generating the log message
- The message text itself, which may be more than one line long

A few sample lines from a `messages` log file are shown here. Notice that the hostname for all of these messages is `brighton`, the name of someone's computer. Also notice that several different programs have generated the log messages shown here, including the Linux kernel, the `httpd` daemon (the Web server), the sound system, and other programs.

```
Oct 26 06:42:27 brighton kernel: parport0: PC-style at 0x378 [SPP,PS2]
Oct 26 06:42:27 brighton nfs: rpc.statd startup succeeded
Oct 26 06:42:27 brighton kernel: parport0: no IEEE-1284 device present.
Oct 26 06:42:27 brighton kernel: lp0: using parport0 (polling).
Oct 26 06:42:28 brighton nfs: rpc.rquotad startup succeeded
Oct 26 06:42:28 brighton kernel: lp0 out of paper
Oct 26 06:42:28 brighton nfs: rpc.mountd startup succeeded
Oct 26 06:42:29 brighton kernel: Installing knfsd (copyright (C) 1996 ok
Oct 26 06:42:29 brighton nfs: rpc.nfsd startup succeeded
Oct 26 06:42:29 brighton keytable: Loading keymap:
Oct 26 06:42:30 brighton keytable: Loading /usr/lib/kbd/keymaps/i386/qwe
Oct 26 06:42:30 brighton keytable: Loading system font:
Oct 26 06:42:30 brighton rc: Starting keytable succeeded
Oct 26 06:42:30 brighton gpm: gpm startup succeeded
Oct 26 06:44:57 brighton rpc.statd[451]: gethostbyname error for brighto
Oct 26 06:45:01 brighton httpd: Cannot determine local host name.
Oct 26 06:45:01 brighton httpd: Use the ServerName directive to set it
Oct 26 06:45:01 brighton httpd: httpd startup failed
Oct 26 06:45:01 brighton sound: Starting sound configuration:
Oct 26 06:45:01 brighton sound: sound
Oct 26 06:45:01 brighton rc: Starting sound succeeded
Oct 26 06:45:02 brighton PAM_pwdb[582]: (su) session opened for user xfs
Oct 26 06:45:03 brighton PAM_pwdb[582]: (su) session closed for user xfs
Oct 26 06:45:03 brighton xfs: xfs startup succeeded
Oct 26 06:45:04 brighton linuxconf: Linuxconf final setup
Oct 26 06:45:04 brighton rc: Starting linuxconf succeeded
Oct 26 06:45:05 brighton rc: Starting local succeeded
Oct 26 06:54:08 brighton PAM_pwdb[629]: check pass; user unknown
Oct 26 06:54:09 brighton login[629]: FAILED LOGIN 1 FROM (null) FOR
roopt, User not known to the underlying authentication module
Oct 26 06:54:11 brighton PAM_pwdb[629]: (login) session opened for user
```

Right now you don't have to understand everything in the preceding log file. But you should become familiar with the format of each line. As you learn about messages from a specific

program, you can read that program's log messages to learn about what the program is doing on your system.

Not all messages are written to the `messages` log file. Many programs write information for system administrators in the `messages` file and other information to other log files. For instance, the Web server daemon generates information that would be useful to the **Webmaster** (the person who manages the content and functioning of the Web server program); this information is stored in separate log files called `access_log` and `error_log`.

The `syslogd` and `klogd` Daemons

Almost every program on a Linux system uses a set of common functions stored in system libraries. Programs share these libraries, as described in Chapter 10, so that the Linux system can use resources more efficiently. Each **function** in a shared library is a set of computer programming code that completes a certain task for any program that wants to use the function. To use a function in a shared library, a program calls the function, or transfers control to the function until the function completes the requested task. The shared system libraries include a function called **`syslog`**. Any program running on Linux can call this function, passing it a message. The `syslog` function then writes these messages to the `/var/log/messages` file. All of the calls to the `syslog` function are managed by a background program called **`syslogd`**, which stands for *system logging daemon*.

The purpose of `syslogd` is to watch for messages submitted by programs, while another daemon, `klogd`, watches for messages submitted by the Linux kernel. **`klogd`** (or the *kernel logging daemon*) logs kernel messages to the `/var/log/messages` file. Both `klogd` and `syslogd` write messages to the same log file. Figure 11-1 shows how these pieces work together to record log messages.

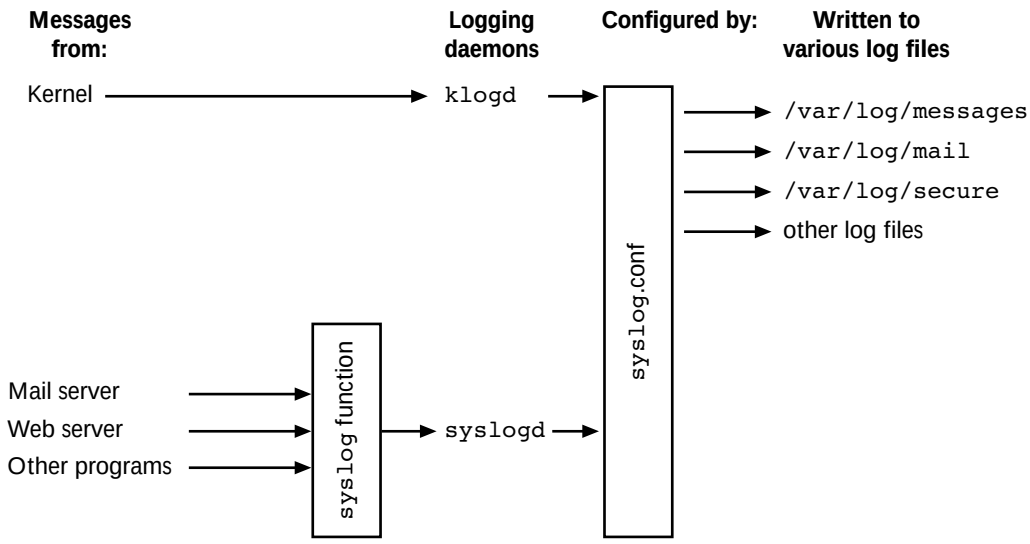

Figure 11-1 How `syslogd` and `klogd` accept messages for the log files

You can see the two logging daemons running on your system by using the `ps` command (which displays processes). The following command begins with a list of all processes (using the `aux` options), then searches the list of processes for those containing the string `logd` (using the `grep` command), and prints matching processes to the screen:

```
$ ps aux | grep logd

root   266  0.0  1.9  1264  592 ?      S    06:42   0:00 syslogd -m 0
root   277  0.0  2.4  1376  752 ?      S    06:42   0:00 klogd
root   690  0.0  1.2  1148  392 tty1   S    07:16   0:00 grep logd
```

In the sample output, you can see that the `START` field (the third from the right, with 6:42 in the first line) contains the time that each daemon was launched. This time matches the time you turned on your Linux system. The system logging daemons are started immediately after Linux is booted to ensure that all activities are logged.

In fact, the messages that are logged as the system is booting can be particularly useful to a student of Linux because they show how the operating system prepares for operation. The start-up messages are also useful for practicing system administrators who need to troubleshoot hardware problems, because the start-up messages show how the Linux kernel initializes each hardware device before use. Any problems related to accessing hardware are listed in the system log file.

The `syslogd` and `klogd` daemons are started by the **system initialization scripts** each time you boot your Linux system. These scripts start all the background processes that maintain Linux as you work. You should never need to start these programs manually after booting Linux.

VIEWING BOOT MESSAGES

When you turn on your computer system, the Linux kernel boots and initializes the computer hardware. The kernel then starts the `init` program, which in turn starts the system logging daemons `syslogd` and `klogd`. The system logging daemon `klogd` is therefore not available when the Linux kernel is initializing the computer hardware. The kernel writes messages to your screen during system start-up (before `klogd` is active). These kernel messages are also stored in a place within the kernel called the kernel ring buffer.

The **kernel ring buffer** is a small area of memory that holds internal kernel messages. If the kernel ring buffer becomes full, the first message written to it (the oldest message) will be discarded as new messages are received. This ensures that the most recent internal kernel messages can be found in the kernel ring buffer.

During system start-up, all of the messages displayed on screen are sent to the kernel ring buffer. However, if many messages are printed during start-up, the first few messages might be discarded to make room for the last few messages.

The exact number of messages or lines stored in the kernel ring buffer varies because each line is a different length. The default kernel ring buffer holds 8192 characters.

The **dmesg** utility shows you the contents of the kernel ring buffer. You can view this information on screen at any time. After you have Linux running, few messages are written to the kernel ring buffer, so it shouldn't change much. Instead, kernel messages are written to the system log file using `klogd`. However, keep in mind that some messages related to device status are still written to the kernel ring buffer after Linux is running. This allows the kernel to store information for viewing by a system administrator even if the system logging functions are not available for some reason. For example, if you insert a new floppy disk or change the PCMCIA cards in a laptop running Linux, the kernel will send a message about the change to the kernel ring buffer.

When you execute the `dmesg` program, use a pipe symbol with the `less` command so you can use the Page Up and Page Down keys to browse the multiple screens of information provided by `dmesg`. The command looks like this:

```
$ dmesg | less
```

Any user can execute the `dmesg` command; you do not need to be logged in as `root`. Some sample output lines from the `dmesg` command are shown here. Notice that the format of each line is very different from the lines in the `/var/log/messages` file. The lines output by `dmesg` sometimes start with a device name, such as `hda` when the first hard disk is detected, but the lines do not have a consistent format. You must read the entire line to see what information it conveys. Some messages fill multiple lines in the sample output that follows.

```
Linux version 2.2.5-15 (root@porky.devel.redhat.com) (gcc version
egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)) #1 Mon Apr 19
22:21:09 EDT 1999
Detected 299946735 Hz processor.
Console: colour VGA+ 80x25
Calibrating delay loop... 598.02 BogoMIPS
Memory: 30760k/32832k available (996k kernel code, 412k reserved,
604k data, 60k init)
VFS: Diskquotas version dquot_6.4.0 initialized
CPU: Intel Mobile Pentium MMX stepping 02
Checking 386/387 coupling... OK, FPU using exception 16 error
reporting.
Checking 'hlt' instruction... OK.
Intel Pentium with F0 0F bug - workaround enabled.
POSIX conformance testing by UNIFIX
PCI: PCI BIOS revision 2.10 entry at 0xfd60a
PCI: Using configuration type 1
PCI: Probing PCI hardware
Linux NET4.0 for Linux 2.2
Based upon Swansea University Computer Society NET3.039
NET4: Unix domain sockets 1.0 for Linux NET4.0.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
```

Because the output of the dmesg program contains much detailed information about how Linux recognizes your system hardware and how it is initialized, consider saving a copy of the dmesg output on a floppy disk for reference in case of hardware problems with Linux. If you are logged in as root, you can copy this information to a standard MS-DOS-format disk using these two commands:

```
# mount -t msdos /dev/fd0 /mnt/floppy
# dmesg > /mnt/floppy/dmesg_text
```

If you ask for Linux technical support from your Linux vendor or from another company, the technical support staff may request the output of the dmesg command to help them understand what is happening on your Linux system.

Because of the importance of this information, some Linux systems store the dmesg output in a file right after the system is started. By doing this, the original boot messages are preserved even if other messages are written to the kernel ring buffer later on. Red Hat Linux stores this information in the file /var/log/dmesg. You can view this file using any text editor or using a command such as this:

```
less /var/log/dmesg
```

CONFIGURING THE messages LOG FILE

The system log file /var/log/messages contains many types of messages from many different programs. Each Linux distribution has a default configuration for the system log files. However, you can specify what information you want to store in this file and what information you want stored in other files. Even if you decide not to change the log file's default configuration, learning about how the log file is configured will help you use the information in the /var/log/messages file.

Both syslogd and klogd are configured using the configuration file named **syslog.conf**, stored in the /etc directory. The /etc/syslog.conf configuration file determines where each type of message from different programs will be logged.

No graphical tools are available to set up the /etc/syslog.conf file. You will rarely need to change this file; but if you do, you must use a text editor.

The Format of syslog.conf

The syslog.conf file is one of the more difficult configuration files to set up. In this section you learn the format of this file and possible values that it can include. A sample line from syslog.conf is shown here. You can refer to this example line as you learn about each element of the syslog.conf syntax.

```
*.info;mail.none;authpriv.none                    /var/log/messages
```

To further help you get started understanding the `syslog.conf` configuration, Figure 11-2 shows the format of each line of the `syslog.conf` file. As with most configuration files, lines that begin with a hash mark (#) are considered comments. In other words, any line beginning with a # character is ignored.

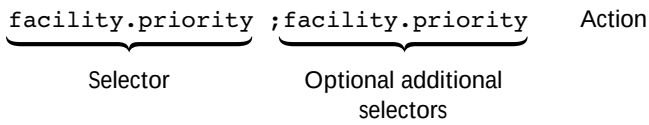

Figure 11-2 The format of each line in the `syslog.conf` file

Each line in the `syslog.conf` file contains two parts:

- A **selector** is a set of code words that selects what events are being logged.
- An **action** is a filename or username that determines either the file in which the message describing an event is written or which user's screen the message appears on. As you will learn later in this chapter, the action can also refer to a file on another (remote) computer.

The selector part of each line is composed of two parts:

- The **facility** is a code word that specifies which type of program is being selected (the category of program providing the log entry).
- The **priority** is a code word that specifies the type of messages being selected for logging.

Each of these items—selector, action, facility, and priority—is described in more detail in the following sections. But seeing an example at this point may be helpful. Consider the sample line shown here:

```
daemon.info                    /var/log/messages
```

The left part of the line contains a selector: `daemon.info`. The facility of this selector is `daemon`. The priority is `info`. Thus, messages from any daemon program with a priority of `info` or higher are selected by these code words. On the right, the line contains an action: `/var/log/messages`. This action is a filename, which specifies that messages selected by the `daemon.info` selector will be written to the file `/var/log/messages`.

The Facilities

When the time comes for a program running on Linux to write a message to the system log file, `/var/log/messages`, it issues a programming call to the `syslog` function. As part of that call, the program must indicate its type, or category. For example, when the `login` program records a message about a user logging into the system, the `login` program specifies that the message is coming from an authentication (security-related) program. The `syslogd` daemon uses this category information to determine where to write the message, based on

the syslog.conf configuration. The actual name of the program (login, in this example), rather than the category (authentication in this example), is written to the log file.

You can configure where messages from each category of Linux program (each facility) are logged by how you set up the syslog.conf configuration file. Table 11-2 lists the different facilities, or types of programs, from which you can select system messages as you set up syslog.conf for system logging.

Table 11-2 Code Words Used to Specify Facilities in syslog.conf

Facility description	Facility name
Messages from user authentication utilities such as login	auth (formerly called security)
Special-purpose (private) user authentication messages	auth-priv
Messages from the cron program (used to control automated, scheduled tasks)	cron
Messages from all standard daemons or servers not otherwise listed by name here	daemon
Kernel messages (through klogd)	kern
Printer server messages	lpr
Mail server messages (from the Mail Transfer Agent)	mail
News server messages	news
Messages about the system logging process itself (such as starting the logging program)	syslog
Messages from programs started by end users	user
Messages from the uucp program (rarely used)	uucp
Eight special-purpose categories that a Linux vendor or programmer can define for specific needs not covered by the other categories	local0 through local7

You use the code words in the second column of Table 11-2 to specify which program's messages you are selecting on each line of the syslog.conf file. Remember, the facility is part of a selector that defines the messages for which you are configuring an action. For example, you might include the auth facility on a line in the syslog.conf file. Whatever action you include on the same line in syslog.conf will apply to the login program and other programs that are categorized as related to authentication (and thus specify the auth facility when logging their messages). If you indicate the mail facility on a line in syslog.conf, the action on that line will apply to the mail server and other programs related to processing electronic mail.

In many cases, multiple programs use the same facility. The daemon facility, in particular, is used by many programs.

The Priorities

All programs running on Linux generate different types of messages. Some messages are simply informational—about how the program is using system resources, for example. Other messages indicate a potential problem. Still other messages indicate a serious or critical problem that will corrupt data or shut down the program. Each program can generate messages with different priorities, depending on the seriousness of the event. You can configure your system so messages of different priorities are logged in different ways.

Table 11-3 shows the different priorities available, from lowest to highest, as you configure the system log files. A priority defines the seriousness of a message. As you would expect, the more serious messages are considered higher priority messages, with `emerge` being the highest priority.

Table 11-3 Message Priorities Supported by System Logging

Priority description	Priority name
Debugging messages used by programmers or those testing how a program works	`debug`
Informational messages about what a program is doing	`info`
Information about noteworthy events occurring as a program executes	`notice`
Warnings about potential problems with a program	`warning` (formerly called `warn`)
Notices about errors occurring within a program	`err` (formerly called `error`)
Critical error messages that will likely cause a program to shut down	`crit`
Error messages that will cause a program to shut down and may also affect other programs	`alert`
Messages about events serious enough to potentially crash the system	`emerg` (formerly called `panic`)

As a software developer writes a program, the developer decides which events are associated with which priority levels. For example, the developer might design a program so that a certain event generates a message with the priority of `warning`. Another programmer might decide that the same event would generate a message with the priority of `notice`. Thus, the programs themselves determine what facility they pertain to and what priority individual events or messages should have. As a system administrator, you simply determine where messages are logged based on their facility and priority.

The Actions

Once you set up a selector (consisting of a facility and a priority), you can assign an action to that selector. This action will determine what `syslogd` and `klogd` do with the messages defined by the selector—either write the messages to a file or display them on the screen of a user who is logged in to the system. Normally the action is simply the name of a local file.

Messages matching the selector are written to the file that you indicate by the `syslogd` or `klogd` daemon. The possible actions are listed here:

- Write the message to a regular file using the given filename.
- Write the message to the terminal indicated. This can be a standard virtual terminal name, from `/dev/tty1` to `/dev/tty6`, or the console device, `/dev/console`.
- Write the message to the screen of any users who are logged in, from a given list of users. For example, if the action is `root,lsnow`, the messages in the selector will be written to the screen of users `root` and `lsnow` if they are logged in.
- Write the message to the log file on a remote system. This is done using the symbol @ in the action. For example, you could specify the action as `@incline.xyz.com` to send log messages for the given selector to the `syslogd` daemon running on the system named `incline.xyz.com`.

Although it's wise to write all messages to a log file, it's also a good idea to send critical messages to the screen or to certain users. This can ensure that the `root` user, or the regular user account of the system administrator, is immediately informed when a very serious error occurs, such as the root file system running out of hard disk space, or a memory error causing a problem in the Linux kernel.

The option of writing messages to a remote system is useful in several circumstances:

- *To consolidate important messages:* many systems in an organization might send all log messages to a single system so they can be archived and studied as a group.
- *To safeguard information on a failed system:* when a system crashes because of a hardware failure, any system log files stored on the hard disk can be damaged, or at least rendered inaccessible until the system is restored. By storing log files on another system, you ensure that these messages can be reviewed even after the system that generated them fails.
- *To enhance security:* storing log files remotely makes it more difficult for intruders to delete records of their activities that may be stored in system log files. Remote log files can thus improve security management.

A network connection between two computers must be available before one computer can store log messages on another computer.

Setting up `syslog.conf`

Now that you understand something about facilities, priorities, and actions, you can review the default `syslog.conf` file on your Linux system to determine how messages are logged. You can also modify that configuration if necessary to suit your own needs.

The following list shows the default `syslog.conf` file from Red Hat Linux. Most of the lines are comments (beginning with a # character). The comments explain the purpose of each configuration line.

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
# kern.*                                                /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none                    /var/log/messages

# The authpriv file has restricted access.
authpriv.*                                       /var/log/secure

# Log all the mail messages in one place.
mail.*                                                /var/log/maillog

# Everybody gets emergency messages, plus log them on another
# machine.
*.emerg                                                *

# Save mail and news errors of level err and higher in a
# special file.
uucp,news.crit                                        /var/log/spooler

# Save boot messages also to boot.log
local7.*                                          /var/log/boot.log
```

As mentioned previously, each selector includes a facility and a priority. As you refer to the preceding sample file, note the following points about formatting the selector and action in `syslog.conf`.

- The asterisk symbol (*) can be used to indicate "all," either for the facility or for the priority. For example, the sample file contains the selector `*.emerg`, which selects all `emerg` priority messages of any facility (coming from any program).
- When an asterisk (*) is used in the action field, it refers to users on the system, meaning that all users who are logged in will receive the message indicated in the selector for that line. The `*.emerg` selector in the sample file above uses * as the action field.
- The facility and the priority are separated by a period. The left side of each line in the sample file contains two items separated by a period.
- The keyword `none` is used as a priority to exclude all messages matching a certain facility.
- Multiple selectors can be included on the same configuration line (thus applying the same action to multiple selectors) by separating selectors with a semicolon (;). For example, one of the lines in the sample file contains `*.info;mail.none;authpriv.none`. Thus three selectors are given at one time.

- Multiple facilities or priorities can be selected at the same time using a comma-separated list. For example, `uucp,news.crit` appears in the sample file, indicating that messages of priority `crit` for either the `uucp` or `news` facility are being configured.
- Whenever a priority is specified in a selector, all messages of that priority and all *higher* priorities (more serious problems) are included in the configuration.
- The same messages can be logged to more than one place by including the same selector on multiple lines with different actions. For example, critical kernel messages might be displayed on the `/dev/console` device and also logged to a remote machine for later analysis. Configuration lines in `syslog.conf` do not override previous configuration lines; rather, each action configured by a line in `syslog.conf` adds to everything already configured in `syslog.conf`.

In addition to these basic rules of syntax for `syslog.conf`, several special symbols are also useful, as described in the following paragraphs. These symbols are not employed very often, but you need to understand them so that you can interpret any log configuration files you might encounter.

When a file is specified as the action (so that messages are written to the file), a hyphen (-) can be added before the filename to indicate that the file should not be accessed in sync mode each time a message is written to it. When a file is accessed in **sync mode**, all information cached in memory is written to the hard drive so that no data will be lost in case of a system crash. Normally, all log files are *synced* as each message is written. This ensures that no log messages are lost if the system crashes suddenly, but it also degrades performance by not taking advantage of the memory caching that Linux provides for optimized hard disk writing. You can choose to improve performance and increase the risk of losing log information by including the hyphen in the action field, for example:

```
*.info;mail.none;authpriv.none                -/var/log/messages
```

As stated previously, including a priority in the selector selects messages of that priority and all higher priorities. You can use the equal sign (=) to specify a priority without including all higher priorities. For example, using the selector `*.=crit` selects all messages from any program (all facilities) that have the priority `crit`, but not the higher priorities `alert` and `emerg`. (These should be configured on a separate line, however.)

You can use an exclamation point to exclude all priorities above a certain level. For example, the following combines these two selectors: `kern.info;kern.!err`. This configuration line selects all messages from the kernel (those with a facility of `kern`) above the priority of `info`, except those with a priority of `err` or higher. In effect, this selects messages of priority `info`, `notice`, and `warning`.

The equal sign and the exclamation point can be used together to exclude a single priority instead of excluding everything above that priority. For example, consider two selectors similar to the example in the previous paragraph: `kern.info;kern.!=err`. These selectors configure an action for all kernel messages of priority `info` and higher, excluding `err` messages, but including `crit` and `emerg` messages (the priorities above `err`).

Few system administrators will need the flexibility provided by using the = and ! symbols within the `syslog.conf` file, but they are available to make your configurations as precise as you decide they need to be.

Table 11-4 describes the effect of several sample configuration lines in `syslog.conf`. Some of these lines are taken from the sample file shown previously; others are taken from the `syslog.conf` file on other Linux systems.

Table 11-4 Sample `syslog.conf` Configuration Lines

Sample configuration line	Effect of the configuration line
`#kern.*  /dev/console`	None; this line begins with a comment character.
`kern.*  /dev/console`	Log all kernel messages (of any priority) to the console (the computer screen).
`*.info;mail.none;authpriv.none /var/log/messages`	Log all messages from any facility with a priority of `info` or higher to the file `/var/log/messages`. But exclude all messages with a facility of `mail` or `authpriv`, no matter what priority.
`authpriv.*  /var/log/secure`	Log all messages from the `authpriv` facility to the file `/var/log/secure`.
`uucp,news.crit  /var/log/spooler`	Write any messages of priority `crit` or higher for the facilities `uucp` or `news` to the file `/var/log/spooler`.
`*.emerg  *`	Display any messages with a priority of `emerg` on the screen of all users who are logged in.
`mail.*  /var/log/maillog`	Log all messages from the `mail` facility to the file `/var/log/maillog`.
`*.emerg  @loghost`	Send all messages of priority `emerg` (from all facilities) to the `syslogd` daemon running on the computer named `loghost`.

You can view additional examples of `syslog.conf` lines in the manual page for `syslog.conf`. (Enter the command `man syslog.conf` to view this online documentation.) Be aware that the manual page includes instructions for all UNIX systems, so the example directories used to store log files will not match everything on your Linux system. The examples, however, are still instructive.

Restarting the System Logging Daemons

After changing the `syslog.conf` configuration file, you must tell `syslogd` and `klogd` to reread the configuration file, so that your changes to the file are implemented on your system. Rather than stop the logging daemons and miss some events that need to be logged, you can send a signal using the `kill` command. Remember from Chapter 8, the `kill` command doesn't always end a program. You can send a program different types of signals using the `kill` command. Some signals end the program; many do not. The signal that you

send with kill (as described in this section) tells the logging daemons to reread the syslog.conf configuration file.

In order to send a signal using kill, you must know the process ID (PID) number for the logging daemons. Several important Linux programs store their PID in a file for occasions when you need to send a signal to the process. The following command shows you the PID of syslogd:

```
cat /var/run/syslogd.pid
```

The following command shows you the PID of klogd:

```
cat /var/run/klogd.pid
```

By inserting the value returned by these commands (the PID of the corresponding logging daemons), you can use the kill command to send a **SIGHUP** signal to each daemon. This signal tells the daemon to reread its configuration files. To send this signal using the kill command, you can use single backward quotation marks to execute the cat command and insert the resulting text as a parameter for the kill command. (Be sure to use single backward quotation marks rather than forward marks.) To summarize, the following command will cause syslogd to reread the syslog.conf configuration file:

```
kill -HUP `cat /var/run/syslogd.pid`
```

Similarly, you can have klogd reread the configuration file using this command:

```
kill -HUP `cat /var/run/klogd.pid`
```


Another acceptable method of restarting the logging daemons is to use the killall command with the name of the daemon, as these two commands show:

```
killall -HUP syslogd
killall -HUP klogd
```

You will have other occasions when you need to send the SIGHUP signal to the system logging daemons as you learn about rotating log files later in this chapter.

Using the logger Utility

The **logger** utility lets you send a message to the syslog function, just as programs do. You can use the logger utility from a command line or from a script file. As you will learn in Chapter 12, a **script** is a collection of commands that functions as a macro, executing commands as if you had executed them on the command line. Once you become proficient at creating scripts, you might want to log events to the system log files.

You can use the logger command with only a message. For example, suppose you created a script to compress files automatically. The script could include a simple logger command like this:

```
logger Compression utility started
```

This would log the message using a default selector of user.notice. Thus, the message would be logged wherever the syslog.conf file had configured messages matching that

selector. (This would normally be in `/var/log/messages`.) Because no additional information is specified, the username of the user running the script is included in the log file as the program name providing the log message. The resulting log entry would look something like this (with the timestamp, machine name, and username varying):

```
Oct 26 11:42:25 brighton nwells: Compression utility started
```

You can also specify other selectors with the `logger` command. For example, to log a message to the `mail` facility with a priority of `info` and the name of the compression script as part of the log file, use this command:

```
logger -p mail.info -t compress Mail folders compressed
```

This would result in a message like the following being written immediately to the log file specified in `syslog.conf` for `mail.info` messages. (The date and machine name vary according to which system is used to execute the `logger` command.)

```
Oct 26 11:46:13 brighton compress: Mail folders compressed
```

MAINTAINING LOG FILES

Ordinarily, your Linux system should require little day-to-day maintenance. Although systems with a large number of users may require that you monitor hard disk and memory usage, most parts of Linux take care of themselves. Log files, however, require and deserve some extra attention. This is true for two reasons:

- Log files contain a valuable record of what has occurred on your Linux system. The information in the log files can be used to check for problems, watch for intruders, and compute statistics about your system.
- Depending on how you have set up `syslog.conf`, log entries can create very large log files. Over time on a busy system, these files will fill up your hard disk.

Checking Log Files for Problems

A system administrator should regularly check log files for indications of problems. By reviewing log files and locating problems before they become critical, you can save a great deal of time and expense troubleshooting and repairing problems that have resulted in security problems, program failure, or even crashed systems.

The busier your system is and the more users with access to the system, the more important the log files will become to your work as a system administrator. By reviewing log files regularly, you will become accustomed to what is normal and what is unexpected. Table 11-5 lists some sample log file entries, along with some possible interpretations. For the sake of brevity, only the program name and message text are shown in the table; the timestamp and computer hostname have been removed from the log entries.

Table 11-5 Interpreting Sample Log File Entries

Sample log entry	System administrator considerations
`login: FAILED LOGIN 3 FROM (null) for nwells, Authentication failure`	Someone has tried to log in as user `nwells` and entered the wrong password three times in a row. If this happens repeatedly in a short period of time, someone may be trying to break in using that user account.
`login: ROOT LOGIN ON tty1`	Someone has logged in as `root`, but the time-stamp (not shown here) indicates that the login occurred at 2 a.m. If no one is expected to be working at that time, an intruder may have access to the system.
`syslogd 1.3-3: restart`	The `syslogd` daemon was restarted. If you did not do this as system administrator, someone may have changed the logging configuration to try to circumvent a security check or cover a security break-in.
`kernel: eth0: NE2000 Compatible: port 0x300, irg 5, hw_addr 00:E0:98:05:77:B2`	The kernel successfully located the Ethernet card as the system booted. The parameters used to access the card are shown in the kernel log message.
`named[339]: Ready to answer queries`	The DNS server has successfully started and is able to respond to requests from clients to resolve domain names to IP addresses.
`modprobe: can't locate module block-major-48`	The `modprobe` command was unable to initialize a device. Some device on the system may not be configured properly.
`kernel: cdrom: open failed`	A user has tried to mount or access the CD-ROM device and either used an incorrect `mount` command or has made some other mistake. The user may need instruction in using the CD-ROM device.
`--MARK--`	The `syslogd` program has inserted a marker to indicate that a fixed amount of time has passed (20 minutes by default). This helps you determine how many messages are written to the log file in each period, but not all systems use this feature. (Red Hat, for instance, does not.)

You can use standard Linux tools like the `grep` utility to search for lines in the log files. For example, to search for all lines in the `/var/log/messages` file containing the program name `login:`, use this command:

```
grep login: /var/log/messages
```

You can also use special log file management utilities that watch your log files for certain conditions that you specify. These utilities notify you (usually via e-mail) about irregularities or potential problems in the log files. You can then take corrective action.

Numerous log analysis utilities are available for free download from sites like LinuxBerg. (Visit *http://xmission.linuxberg.com/conhtml/adm_log.html*.) Example programs available at this site include **logscanner** and **LogWatch**. Both allow a system administrator to configure specific items to watch for in system log files based on criteria such as username, security level, and time frame of the event.

Rotating Log Files

With so many programs writing messages, the log files on your system can become very large. Obviously, the busier your system is, the more messages will be written to your log files in a given period of time. But on every system there is a limit to how large the log files can become without using an undue amount of disk space. This varies depending on your system. If you are working on an older system with only 500 MB of disk space, you might not want to use 1 MB of disk space for log files. Conversely, if you are running a large Linux system for many users and have 50 GB of disk storage space, dedicating 500 MB (1%) of the disk to log files that help you track how the system is operating might be perfectly acceptable.

Over time, however, all log files become too large. Part of every system administrator's job is to regularly rotate the log files so they can be used appropriately. The process of **rotating log files** might mean any of the following:

- Discarding old log files to provide disk space for new log information
- Compressing log files and storing them on an archive medium as a long-term record of system activity
- Renaming and compressing the log files so they can be studied at some future time

You don't need to save most log files forever. It's common to use a rotation system for log files, so that at any time, the past few days, weeks, or months worth of log files are available for review, each in separate files. Your particular circumstances dictate whether you use a separate file for each day, each week, or each month, and how many of those files you maintain. Log files are normally moved to another directory and often to another file system (another hard disk or hard disk partition) to free up space on the root partition.

For example, suppose you want to maintain four weeks worth of archived data for the `/var/log/messages` log. You plan to maintain this data as four files: `week1`, `week2`, `week3`, and `week4`, with the newest data being written to the `/var/log/messages` file by `syslogd`. Each Monday morning you rotate the log files. The basic process for rotating log files looks like this:

1. Rename all old log rotation files. In this example, `week4` is discarded, `week3` is renamed to `week4`, `week2` is renamed to `week3`, and `week1` is renamed to `week2`.
2. Rename the `/var/log/messages` file to `week1`.
3. Create a new file named `/var/log/messages` (initially, it is empty).
4. Send the `syslogd` and `klogd` programs a SIGHUP signal so that they begin to use the new `/var/log/messages` file.

The last two steps might be confusing at first glance. Why do you need to create a new file named `/var/log/messages` and then issue a SIGHUP signal? The reason revolves around how Linux accesses files. As the `syslogd` and `klogd` programs are launched, they begin by locating the log files they will write to. The `/var/log/messages` file is identified by a number called an inode (which you may recall learning about in Chapter 9). If you change the name of the file `/var/log/messages`, the inode that refers to that data remains the same, so the two logging daemons continue writing data to the file that you have renamed `week1`. By creating a new file called `/var/log/messages` and sending a SIGHUP signal, the logging daemons look for the `/var/log/messages` file again by name, obtain the inode for the new (empty) file you have created, and begin writing messages to that new file.

The commands required to accomplish the steps given above might look like the following, depending on where you are storing your log files. (The numeric endings on the files shown in this example are commonly used to designate archived log files.)

```
cd /archive
rm -f messages.4
mv messages.3 messages.4
mv messages.2 messages.3
mv messages.1 messages.2
mv /var/log/messages /archive/messages.1
touch /var/log/messages
killall -HUP syslogd
killall -HUP klogd
```


Because a typical Linux system includes many log files—for the Web server, e-mail, security messages, kernel messages, and so forth—it is common to rotate many different log files each day, week, or month, depending on how rapidly the various log files are growing. For example, if you are running a busy Web server, the Web access log file (`/var/log/httpd/access_log`) might need to be archived each night, while on the same system the `/var/log/messages` file is rotated weekly or every two weeks.

In Chapter 12 you will learn how to create a script to automate commands like these. One very helpful program to include in your scripts is the `logrotate` command described in the next section.

Using the `logrotate` Utility

Red Hat Linux provides a utility called **`logrotate`** that you can use to help you rotate all of the log files on your system. To use `logrotate`, you must set up a configuration file that the `logrotate` command uses. The `logrotate` command is normally executed automatically on a regular basis (using methods described in Chapter 12). However, using this utility is much easier than creating and maintaining separate lists of commands for each of your system's log files.

The configuration file for `logrotate` can be stored anywhere you choose. You must provide the name of the configuration file when you launch the `logrotate` command. A sample configuration file is shown here. This file only manages rotation of two log files: `/var/log/messages` and `/var/log/httpd/access_log` (for the Web server).

```
# sample logrotate configuration file
errors root@mycompany.com
compress
/var/log/messages {
      rotate 4
      weekly
      postrotate
            killall -HUP syslogd
      endscript
}
/var/log/httpd/access_log {
      rotate 10
      size=10M
      mail webmaster@mycompany.com
      postrotate
            killall -HUP httpd
      endscript
}
```

The following list explains this sample configuration file:

- Any error messages generated during the log rotation process are e-mailed to `root@mycompany.com`.
- All archived log files are compressed using the `gzip` utility.
- The `/var/log/messages` file is rotated weekly (assuming that the `logrotate` command is executed at least that often). Four old files are saved (as in the previous manual example). After each rotation (once per week) the `killall` command is used to begin using the new log files (again as in the manual example given previously).
- The Web server log file `/var/log/httpd/access_log` is rotated whenever its size exceeds 10 MB. A message is sent to `webmaster@mycompany.com` to confirm each log rotation operation. As with the `/var/log/messages` configuration, the `killall` command is used to make all `httpd` Web server daemons use the new log files after rotation.

Supposing you had saved the above configuration file as `/archive/logrotate.conf`, you would execute the `logrotate` command using this syntax:

```
logrotate /archive/logrotate.conf
```

The `logrotate` utility provides additional functionality that you can learn about in the online manual page for the utility (using the command `man logrotate`).

The `logrotate` command is not available on all Linux systems.

CHAPTER SUMMARY

- Log files are an important part of system maintenance because they track all important activities occurring on a Linux system. They can be used for tracking down hardware problems, computing statistics, and identifying security dangers.
- The `syslogd` and `klogd` daemons store log messages based on the contents of the `/etc/syslog.conf` configuration file. All programs can use this mechanism to save data to log files. You can also use the `logger` utility to write messages to the log file from any command-line interface. The main system message file is `/var/log/messages`.
- The `syslog.conf` configuration file defines categories of messages (called facilities), as well as priorities for each message. Different priorities define how serious a message is. A facility and a priority together make up a selector. In the configuration file, you assign actions to selectors to determine how information is logged on your Linux system.
- Because log files can grow rapidly, you must maintain them regularly. This is normally done by rotating log files, saving older information for review as appropriate. Some specialized utilities such as `logrotate` are available to help with this task.

KEY TERMS

action — A field in the `syslog.conf` configuration file that determines what to do with messages matching the selector on that line.

daemon — A background process that does not display status information on the screen. Instead, daemons normally write information to log files.

`dmesg` — Program that displays the contents of the kernel ring buffer. This buffer normally contains hardware configuration data generated during system start-up.

facility — A category assigned to a system message, identifying the type of program providing the message.

function — A set of computer programming code that completes a certain task for a program.

kernel ring buffer — A small area of memory that holds internal kernel messages. These messages can be viewed using the `dmesg` utility.

`klogd` — A background program (or daemon) used to log kernel messages according to the configuration given in the `syslog.conf` configuration file.

log file — File that contains detailed records of activity on a Linux system.

`logger` — A program that lets you send a message to the `syslog` function. Such messages are written to the log files according to the configuration in `syslog.conf`.

`logrotate` — A program that manages the rotation of multiple log files at regular intervals according to a configuration file created by the system administrator.

logscanner — A log analysis program available for download at *http://xmission.linuxberg.com/conhtml/adm_log.html*.

LogWatch — A log analysis program available for download at *http://xmission.linuxberg.com/conhtml/adm_log.html*.

message — A description of what is happening within a program.

`messages` — The main system log file in Linux, usually stored in the directory `/var/log`.

priority — A number indicating the severity of a message submitted for logging. Log configurations are often based on the priority of incoming messages.

rotating log files — The process of moving existing log files to another filename and location for archiving or review. Rotating log files frees hard disk space for new log messages.

script — A collection of commands that functions like a macro, executing commands as if you had executed them on the command line.

selector — A field in the `syslog.conf` file that determines what events are being logged. The selector is composed of a facility and a priority.

SIGHUP — A signal sent to a logging daemon to instruct the daemon to reread its configuration files and the log file it writes to.

sync mode — An option assigned to log files by which the data written to the log file is immediately written to the hard disk rather than being cached in memory to improve system performance.

`syslog` — A programming function used by Linux programs to submit messages for logging. The `syslog` function interacts with the `syslogd` daemon to write messages according to the `syslog.conf` file.

`syslog.conf` — The configuration file used to control how and where messages are logged by `syslogd` and `klogd`.

`syslogd` — The background program (or daemon) that manages all of the calls to the `syslog` function, writing log messages according to the `syslog.conf` configuration.

system initialization scripts — Instructions executed each time you boot your Linux system.

timestamp — The date and time when an event being logged occurred.

Webmaster — The person who manages the content and functioning of a Web server program running on Linux.

REVIEW QUESTIONS

1. All messages generated by standard Linux services and the kernel are logged in the `/var/log/messages` file. True or False?
2. Log files are generally *not* used for which of the following tasks?
   a. Watching for security problems
   b. Calculating system usage statistics

c. Calculating memory usage for applications

d. Determining the cause of system failures

3. Given the log entry,

```
Oct 26 06:45:01 brighton httpd: Cannot determine local host
name
```

the word `httpd` refers to which of the following?

a. The system name on which the event being logged occurred

b. The program that generated the event being logged

c. The daemon handling the logging of the event

d. The configuration file used to control logging of this event

4. Explain the differences between the `syslogd` and the `klogd` logging daemons.

5. A standard `/var/log/messages` log entry contains all of the following *except*:

a. The hostname of the computer on which the event occurred

b. The name of the daemon that processes the `syslog` function call

c. A timestamp showing when the event occurred

d. Message text describing the event

6. The `syslogd` and `klogd` daemons must be started by the `root` user as soon as the system is booted and `root` logs in. True or False?

7. The kernel ring buffer is:

a. A holding area for messages generated by `klogd` before writing them to `/var/log/messages`

b. An internal storage area used by the kernel for certain types of messages

c. A disk cache area used by the kernel

d. A symbolic link to the kernel source code

8. The `syslogd` and `klogd` logging daemons depend upon which configuration file?

a. `logrotate.conf`

b. `syslog.conf`

c. They are internally configured and use no configuration file

d. The `syslog` function called by individual applications

9. A configuration pair consisting of a facility and a priority is called:

a. An action

b. The timestamp

c. A selector

d. A SIGHUP signal

10. Name four types of actions that can be associated with a selector when configuring how messages are logged.
11. The `dmesg` utility is used to do which of the following?
    a. View the kernel ring buffer contents
    b. Sync the log files, writing them to disk
    c. View the most recently logged device messages
    d. Configure events logged to the `messages` file
12. Priorities associated with logged events determine how quickly the message is logged. True or False?
13. The selector `*.info` will log the following:
    a. Messages from all facilities with a priority of `info` or higher
    b. Messages without a facility assigned with a priority of `info`
    c. Messages with a facility of `info` and any priority
    d. Messages from the `info` command that will be posted on the screens of all users who are logged into the system
14. Which of the following is not a valid facility name?
    a. `auth`
    b. `httpd`
    c. `user`
    d. `mail`
15. This configuration line

    ```
    *.emerg    @brighton
    ```

    will cause which of the following to occur?
    a. All messages with a facility of `emerg` are logged to a file matching the system name (the hostname).
    b. All messages with a priority of `emerg` or lower are logged to the file configured as an alias to `brighton`.
    c. All messages with a priority `critical` are sent to the machine named `brighton` for logging.
    d. All messages of any priority but `emerg` are displayed on the screen of user `brighton`, if that user is logged in.
16. Describe briefly the special configuration characters !, =, and - as used in the `syslog.conf` file.
17. An asterisk in the action field of `syslog.conf` is equivalent to an asterisk in the selector field of `syslog.conf`. True or False?

18. This configuration line

```
*.info:mail,news.none:authpriv.none      -/var/log/messages
```

is invalid because:

a. A colon cannot be used to separate multiple selectors.

b. Each selector can only include one facility.

c. The `none` keyword cannot be used as a priority.

d. The hyphen can only be used in the action field when associated with a set of usernames.

19. Describe why the system logging daemons must be restarted in order to access new configuration files.

20. The command `cat /var/run/syslogd.pid` does which of the following?

a. Causes the `syslogd` daemon to reread its configuration file

b. Prints the PID of the currently running `syslogd` daemon

c. Sends a SIGHUP signal to the logging configuration file

d. Prints out the most recent logging messages

21. The `logger` utility does which of the following?

a. Sends messages to the `syslog` function for logging according to the `syslog.conf` file

b. Writes messages to `/var/log/messages`

c. Rotates log files according to a predetermined configuration

d. Restarts the logging daemons with a SIGHUP signal

22. Log file messages can be analyzed using standalone specialized tools or a basic text editor at the command line. True or False?

23. Describe how the `logrotate` command is configured.

24. Which of the following is a valid reason to rotate your log files?

a. Leaving them open for long periods can cause file corruption.

b. The files become too large to store on the `root` partition.

c. System administrators cannot study live log files.

d. Security-minded individuals feel rotated log files are safer.

25. If you saw the message

```
login: FAILED LOGIN 3 FROM (null) for nwells, Authentication
failure
```

you might reasonably assume any of the following *except*:

a. Someone is trying to break into your system using the `nwells` account.

b. User `nwells` has forgotten his password.

c. The `login` program has become corrupted.

d. A user on your system is trying to break into the files owned by `nwells`.

HANDS-ON PROJECTS

Project 11-1

In this activity you watch the system log file as new messages are written to it by the `syslogd` daemon. To complete this activity you need an installed and working Linux system with a graphical interface.

1. Log in to Linux as `root` and start the graphical environment.
2. Open two terminal emulator windows (command-line windows).
3. In one of the windows, enter **`tail -f /var/log/messages`** to display the last 15 lines of the system log file, updating the display every few seconds.
4. In the second window, enter **`killall -HUP syslogd`** to restart the `syslogd` daemon. Notice that a message is added to the first window stating that the `syslogd` program was restarted.
5. In the second window, enter **`killall -HUP httpd`** to restart any Web server daemons running on your system. Notice the messages that are added to the `messages` file in the first window.
6. Leave the first window open for a few minutes as you work on your system, opening other applications or browsing the Web. Are additional messages written to `/var/log/messages`? Can you interpret the messages?

Project 11-2

In this activity you use the `logger` command to send a message to the `/var/log/messages` system log file. To complete this activity you need an installed and working Linux system with a graphical interface.

1. Log in to Linux as `root` and start the graphical environment.
2. Open two terminal emulator windows (command-line windows).

3. In one of the windows, enter **`tail -f /var/log/messages`** to display the last 15 lines of the system log file, updating the display every few seconds.
4. In the second window, enter **`logger -p user.info -t TESTING This is a logging test`**.
5. Notice the message that is added to the `/var/log/messages` file shown in the first window.
6. Using the facility and priority names you learned in the chapter, try sending one or two other messages using the `logger` program. In particular, try sending a message with the priority `emerg`. For example, you might use this command: **`logger -p user.emerg -t TESTING Emergency message test`**. What do you notice about how this command is treated compared to the other `logger` commands you entered? Can you explain why, based on the information in the `syslog.conf` file?

Project 11-3

In this activity you explore the contents of the kernel ring buffer using the `dmesg` command. To complete this activity, you need an installed and working Linux system.

1. Log in to Linux.
2. Enter the command **`dmesg | less`** to display information on the screen.
3. Use the Up and Down arrow keys and the Page Up and Page Down keys to scroll through the information. How would you describe this information? How does the format compare with that of the `/var/log/messages` file you saw in the previous two projects?
4. Describe some of the Linux device names that you recognize in the output of `dmesg` that you are viewing. Can you see an Ethernet card, a SCSI device, a video card, or a CD-ROM drive? (Not all devices are part of every Linux system.)
5. Press the **q** key to exit the `less` command.
6. Enter **`dmesg | grep hda`** to search the output of `dmesg` for lines that include information about your first IDE hard disk (device `/dev/hda`).

CASE PROJECTS

1. The main Linux file server used by employees at Marcus Financial Group is accessed heavily every day by about 50 people who share applications, send files for printing, and store their documents on the server. The IT management is considering making the file server into a gateway for the Internet so that employees can also browse the Web from their desktops. You have many concerns about this, ranging from the increased load on the network to the security of your Linux server. What precautions might you take with the log files stored on the Linux server as you create an Internet gateway? Will remote storage be part of your solution? Do you anticipate needing additional hardware resources because of the log files? What tools might you try using to alleviate your security fears? Would your efforts in this project also be beneficial to all the employees before the creation of the Internet gateway? How?
2. After the Internet gateway has been up and running for two weeks, you receive a midnight call from an employee working late. You learn that the Linux server has "crashed," or stopped responding to the employee's Web browser requests. Assuming that the server did not have a hard disk failure (thus the log files are still readable), how could you use the log files to help determine the cause of the problem? What effect would your previous preparations (discussed in Question 1) have on your ability to track down the problem using the log file information? Does this problem lead you to change the way the log files are managed or configured?
3. As you implement the revised plan that you conceived in Question 2, do you see any reason to use the `logger` command in your administration scripts? What circumstances might make that utility useful to you at Marcus Financial Group? Will you use the `logrotate` command if it is available on the version of Linux that you are using?